AN EFFICIENT ALGORITHM FOR RELATIONSHIP INFERENCE

A Thesis

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Computer Science

by

Ramya S Babu

May 2018

ABSTRACT

Correctly inferring relatedness among samples is essential for genetic analysis. It can be helpful for preventing false signals in genetic association studies and finding relatives in forensic genetics. However, relatedness among samples is not always obtained when collecting sample data; in most cases, the relatedness is unknown and needs to be determined. Here, we develop an algorithm to infer relatedness among samples that we aim to be more efficient than current related approaches such as PLINK. Our approach is based on finding stretches of shared alleles across windows of moderate length (3-5 centiMorgans) in the genome. With this information, the algorithm infers the degree of relatedness up to the third degree from the number of windows that are similar between a pair of individuals.

BIOGRAPHICAL SKETCH

Ramya Babu graduated in May 2016 from the University of Illinois at Urbana-Champaign (UIUC) with a Bachelor of Science in Bioengineering and a minor in Computer Science, with a specialization in Computational Systems and Biology. At UIUC, she involved herself with research in the MatMed laboratory for Materials in Medicine by working with Dr. Dipanjan Pan on a project to create 3D printed cardiovascular stents for personalized medicine. This research has been published in the journal of *Advanced Healthcare Materials.* After her graduation from UIUC, she attended Cornell University to pursue a Master of Science in Computer Science with a minor in Genomics. During her time at Cornell, she has been a teaching assistant for CS 2110: Object-Oriented Programming and Data Structures and CS 1109: Fundamental Programming Concepts. She has also conducted research under the guidance of Dr. Amy Williams on developing an efficient algorithm for relationship inference.

This document is dedicated to all the friends, family, and faculty who have supported me over the past two years.

"DNA is like a computer program
but far, far more advanced than any software ever created."
- Bill Gates

ACKNOWLEDGEMENTS

I would like to thank my research advisor Dr. Amy Williams for her constant guidance and mentoring throughout this project, without whom I undoubtedly would have struggled in making my algorithm as efficient and precise as it is. Dr. Andrew Clark, thank you for serving as the minor field representative on my committee and for providing numerous insightful and valuable comments to improve my research and helping me understand the biological significance of my work. To my labmates, it was an immense pleasure getting to know and work with all of you, thank you for your support and help with my research. To my friends and classmates, you've made my time at Cornell quite enjoyable. To Cornell University, thank you for the many opportunities you've provided me with to expand my skills and pursue my dreams and goals. Lastly, I would like to acknowledge all my friends and family for their constant support, thank you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTORY INFORMATION

1.1 Significance

Genomic data contains valuable information regarding the traits and characteristics that an individual will inherit and possibly transmit to their children. Many genetic variants have been linked to a variety of diseases and phenotypes, including some associations to specific genes.[1, 2] Studying the inheritance patterns of phenotypes among close relatives can provide clues about which genes (or portions thereof) are responsible for specific conditions or traits.[3] Researchers use pedigrees to determine genetic heritability models for traits and disorders.[4] This knowledge about pedigrees and inheritance has the potential to help us better understand diseases and combat them by helping to identify causal genes and potential therapeutic targets. Aside from genetic disease association studies, relatedness inference is also important for reconstructing pedigrees that can be used for recombination and mutation studies.

1.2 Background

Studying genetic information has been of interest throughout the 20th century. The launch of the Human Genome Project with the goal of determining the sequence of base pairs constituting human DNA revolutionized the scientific community in 1990. In the early 2000s, the sequencing of the human genome was declared complete. [5] With its completion and advancing technology, many

projects such as the International HapMap Project [6] and the 1000 Genomes Project [7] began sequencing the DNA of thousands of individuals. In fact, there are now many commercial services that use DNA to trace genealogy for the general public at relatively affordable prices. [8] For such projects and services, since current sample sizes range from the hundreds of thousands to millions of subjects, it is important to be able to analyze the genomic data quickly and efficiently.

With the completion of the human genome project, one of the next steps was to pursue studies to identify genetic variants responsible for specific traits and diseases. The inheritance patterns for traits and diseases can be studied using heritability models. These models can be analyzed using pedigrees, which can be inferred from relationships revealed from analysis of the genomic data.[10] Therefore, knowing a pedigree structure enables correct identification of the mode of inheritance. This knowledge has the potential to help us better understand diseases and ultimately combat them. [4]

Family genetic data is also used for linkage analyses of diseases and quantitative traits. These studies typically assume that the relationships between individuals within families are known with good confidence. Misclassification of relationships can lead to reduced or inappropriately increased evidence for linkage. Therefore, it is important to ensure that the relationship between a given pair of individuals is correct.[9] In situations where the relationship is either unknown or uncertain, genetic analysis can be used to validate presumed relationships. Additionally, analysis of genetic data to infer relationships and relatedness has been used in a wide variety of other applications. In genetic association studies, close relatives must be accounted for to avoid biased genetic

signals and associations.[3] In forensic genetics, analysis of genetic data is used to aid in determining relatives of missing persons or victims of disasters.[12] In population genetic analyses, relationships need to be inferred to avoid bias. However, the reach of relationship inference has now grown outside the scientific community. Relationship inference has made an impact within the industry as well. There are some companies that allow the general public to explore their ancestry and genealogy.[8] Additionally, marriage and inheritance laws are sometimes based on the degree of relatedness between individuals.[11]

There are now many existing genetic analysis tools to infer relationships. However, many of these are not designed to handle large datasets efficiently. Of these pre-existing tools, PLINK, an open-source C/C ++ whole-genome association study tool set, is one of the more efficient methods.[13] With PLINK, large data sets comprising hundreds of thousands of markers genotyped for thousands of individuals can be rapidly manipulated and analyzed in their entirety. A second-generation version of PLINK was released to improve performance and compatibility across platforms. However, there is still room for further development.[14] The aim of this project is to infer relationships more efficiently than PLINK but with comparable accuracy.

CHAPTER 2

METHODS

Our method primarily focuses on improving efficiency for inferring relatedness between a pair of individuals by analyzing the sharing of single alleles across multiple adjacent markers in unphased data. The process of phasing (i.e. inferring haplotypes) is computationally expensive and our approach of allele-sharing avoids this expense. Some other considerations to improve efficiency include less use of branching and use of bitwise operations. The algorithm identifies sets of individuals that share genomic data up to the third degree of relatedness. The pseudocode below in Algorithm 1 gives a general overview of how the algorithm works. The upcoming sections will explain the algorithm in more specific terms and details.

Algorithm 1 Inferring relationship between individuals

```
Require: x is the input file containing the genomic data
function RelationshipInference(x)
    Read in genomic data from x
    Create sets to store which samples carry each allele at each marker
                                  ▷ Analyze genome in windows and superwindows
                                          ▷ A window is a sequence of markers
                                  ▷ A superwindow is a sequence of windows
    for each individual i do:
        for each superwindow sw do:
            Divide sw into w smaller windows
            Find the set of individuals that share alleles with i in each w
            if considering genotyping error then
                Taking the union of the sets for each w gives the set for the sw
            else
                Intersecting the sets for each w gives the set for the sw
            end if
        end for
        Infer relatedness from number of superwindows that share alleles
    end for
end function
```

2.1 Algorithm Details

First, the algorithm parses the input file to obtain the number of individuals and markers in the dataset by using the genetio library. The library constructs a data structure that contains genotype data for each individual. The PLINK format genotype data read by the algorithm is a sequence of 0s, 1s, 2s, and 3s as indicated by the definitions below[13, 14]:

- 0 = homozygous for minor allele
- 1 = missing data
- 2 = heterozygous for both alleles
- 3 = homozygous for major allele

Using this genotype data, we construct sets to indicate which individuals carry each allele at each marker. See Figure 2.1 for an illustration of this data structure. For each marker position, we store two sets to denote which individuals carry the major allele and which individuals carry the minor allele. If the data is heterozygous (2) at a given marker position, then the individual carries both alleles and we place this individual in both the sets for the major and minor alleles. If the data is missing (1) at a given marker position, we do not know whether the individual carries the major or minor allele. Rather than make an assumption towards one of the two alleles, we assume that the individual is equally likely to carry either the major or minor allele and therefore put individuals with missing data into both sets.

With the construction of these sets of allele sharing, we analyze the genome in windows — where a window is determined by a consecutive set of markers.

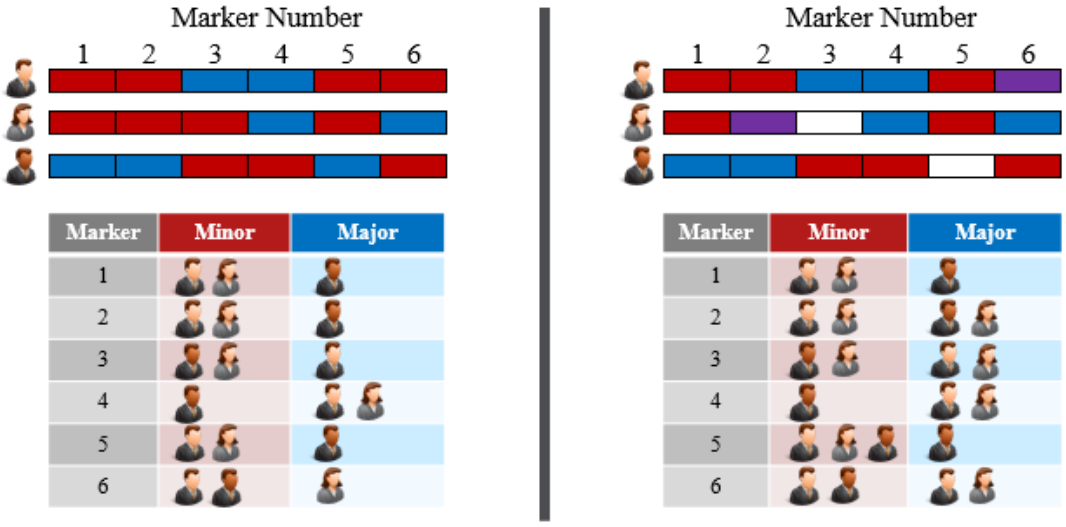

Figure 2.1: Left: Sets of major and minor allele sharing for each marker without any missing data or heterozygousity. Right: Sets of major and minor allele sharing for each marker with both missing data and heterozygousity. Both: Homozygousity for the minor allele is shown in red, homozygousity for the major allele is shown in blue, heterozygousity for both alleles is purple and missing data is white. A representative sample size of three individuals is used in this illustration.

A window is also constrained to have a maximal span in terms of Morgans. An input parameter to the algorithms supplies this constraint. The sequence of markers within a window must have a span (in terms of Morgans) that is less than the maximal span.

We do analysis with respect to a reference individual and obtain the set of individuals that share alleles with this person within a specified window. Algorithm 2 shows how this works. We initially assume that all individuals share alleles with the reference individual. Then, we obtain the set of individuals that share an allele for each marker where the reference individual is homozygous for one of the alleles and use set intersection to ”remove” the individuals that don’t share an allele at a given marker. This will give us the set of individuals that share alleles across all the markers in the window. We consider this

set to be a set of "matching" individuals where a pair of individuals in this set "match" in that they share alleles at every marker in the window. Theoretically, we can use the entire genome as one window and obtain the set of individuals that share alleles with a reference individual in its entirety - though doing so is not practical or useful as almost no relationship types share an allele at every marker. Therefore, we use smaller windows and experiment with the window length to find the size that produces the best performance, as discussed later in the thesis.

Algorithm 2 Obtaining the set of individuals that share alleles in a window

Require: *i* refers to the reference individual
Require: *start* and *end* correspond to the start and end markers of a window

```
1:  function WINDOWSHARING(i, start, end)
2:      Let matchset initially be a set containing all individuals
3:      for each marker m in start...end do
4:          Let g = genotype for individual i at marker m
5:          if g is homozygous for either the major or minor allele then
6:              Let a = allele of homozygosity
7:              Let alleleData = set of individuals for marker m and allele a
8:              matchset = matchset AND alleleData                ▷ bit-wise AND
9:          end if
10:     end for
11:     return matchset
12: end function
```

With window analysis made possible, we further analyzed the genome as "superwindows" and smaller windows. The motivation behind using superwindows is to aid in accounting for genotyping errors. A superwindow is a set of consecutive windows. Figure 2.2 shows an illustration depicting the difference between a window and a superwindow. The number of superwindows and the window length are provided as parameters to the algorithm. Now that we have a function (Algorithm 2) to get the set of the individuals that match a specific sample in a given window, we use that as a helper function to get the

set of individuals that share alleles in each superwindow. The pseudo-code for this is seen in Algorithm 3. Given the number of superwindows, we calculate the average number of markers per superwindow. Now for each superwindow, we iterate over the markers within it to find the smaller windows contained in it. When finding the smaller windows, we define the end point of a window to be the marker that is at most one window's length away from the start position. Once an end point is found, we find the match set for the defined window and then update the starting marker. When iterating over the markers to find windows, if the current marker is on a different chromosome from the starting position, then we end the window. When accounting for genotyping error, we define the set of individuals that share alleles in a superwindow as the set of individuals that share alleles in at least one of the smaller windows within that superwindow. In other words, the union of the match sets for the smaller windows will give the match set for the superwindow. The other approach that does not account for genotyping error uses superwindows that are the intersection of the sets from smaller windows. In the approach that does not account for genotyping error, individuals have to share alleles in the entirety of a superwindow. In this case, the superwindows are just larger windows.

Figure 2.2: Illustration of the difference between a window and a superwindow

Given the data structure that stores the sets for each superwindow, we used Algorithm 4 to obtain the set of individuals that share alleles in a specified number of superwindows. This algorithm works by first initializing the set of individuals that match to be empty. For each individual, we check if it matches in

Algorithm 3 Obtaining match sets across each superwindow

Require: *numsw* refers to the total number of superwindows in the genome
Require: *i* refers to the individual to compare the dataset against
Require: *markers* is the number of markers in the genome
Require: *len* is the maximum length of a window in cM

```
1: function SUPERWINDOWS(i, numSW, markers, len)
2:     Let superwindows be an array of size numsw to contain the set of individuals that match i per superwindow
3:     mpsw = markers / nums;                          ▷ avg. no. markers per sw
4:     start = 0                                       ▷ First window starts at first marker
5:     for each superwindow s in 1...numsw do
6:         Initialize superwindows[s] to be the empty set
7:         for each marker m in 1...mpsw do
8:             dist = distance of m to start
9:             if dist > len then
10:                end = m - 1
11:                wm = WindowSharing(i, start, end)
12:                superwindows[s] OR wm                ▷ bit-wise OR
13:                start = m
14:            else if m == mpsw then
15:                end = m
16:                wm = WindowSharing(i, start, end)
17:                superwindows[s] OR wm                ▷ bit-wise OR
18:                start = m + 1
19:            end if
20:            if m is on different chromosome from start then
21:                start = m
22:            end if
23:        end for
24:    end for
25:    return superwindows
26: end function
```

at least a threshold number of windows. If it matches, then we add it to the set of related individuals.

Algorithm 4 Obtaining matches in at least t windows

Require: *swData* refers to the result of SuperWindows
Require: t represents the threshold number of superwindows to match to
Require: *sw* refers to the total number of superwindows
Require: *numIndivs* refers to the total number of individuals

```
1: function THRESHOLDMATCH(swData, t, sw, numIndivs)
2:     Let tData initially be the empty set.
3:     for each individual i in 1...numIndivs do
4:         Let sum = the number of windows that individual i matches
5:         if sum ≥ t then
6:             Add individual i to tData
7:         end if
8:     end for
9:     return tData
10: end function
```

2.2 Parameters

There are many parameters that the algorithm takes into consideration in order to infer relatedness. This section provides an explanation of the various parameters that are used.

2.2.1 Number of Superwindows and Window Length

The first parameter considered is the number of superwindows to divide the sequence into. Each of the superwindows is then further divided into windows. These windows are determined by a specified window length. The window length is provided to the algorithm in terms of Morgans. A centiMorgan(cM), one hundredth of a Morgan, is a unit of recombination frequency and measures genetic distance. In humans, 1 cM corresponds to roughly about 1 million base pairs. [16] Given that the human genome is approximately 3 billion base pairs, if we have 500 superwindows and a 3 cM window length, there are about 2

windows per superwindow on average.

2.2.2 Threshold values

The algorithm requires threshold values to aid in differentiating between the degrees of relatedness. The threshold values correspond to the number of superwindows where allele-sharing needs to occur in order for a pair of individuals to be inferred as related. The algorithm requires one threshold value for each degree of relatedness that it attempts to identify. On average, first degree relatives share 50% of their DNA, second degree relatives share 25% and third degree relatives share 12.5%.[15]. We used these averages as starting points in choosing the threshold values. We typically run the algorithm in a mode that tolerates errors, so allele-sharing in a superwindow means that there is allele-sharing in at least one window contained within it.

2.3 Testing

At one stage of the algorithm development, we had two versions of the program. One version used branching and the other bypassed use of branching. In a program, branching occurs when it executes different sets of instructions under different conditions. For example, use of if-statements is one form of branching where the evaluation of a boolean statement determines the set of instructions to execute. Our approach used mathematical operations that are guaranteed to have no effect when a given condition is not true and would have the needed effect when the condition is true. We compared the runtime of

the two algorithms at this point to see if it was worthwhile to continue development without the use of branching. We found that branching made little to no difference in the runtime (data not shown.)

After the branching analysis, we completed the algorithm with superwindow matches defined as the intersection of the smaller window matches, i.e. non-error tolerant. At this point, we did some parameter tuning to determine the set of parameters that results with the best performance. Using `ped-sim` [17], we simulated genomic data that is consistent with a specified pedigree structure. Figure 2.3 shows an illustration of the simulated pedigree structure. Though the figure has the sexes specified, `ped-sim` randomizes the sexes of the individuals in distinct copies of the pedigree. We supplied a collection of European-descent individuals to `ped-sim` to use as founder individuals for simulation. [18] We initially simulated 10 families giving a sample size of 380 individuals. To find the set of parameters that yielded the best results for our approach, we varied the following parameters and then obtained accuracy results: the number of superwindows, maximum length of a window, and the threshold requirements for an individual to be a first, second, or third degree relative.

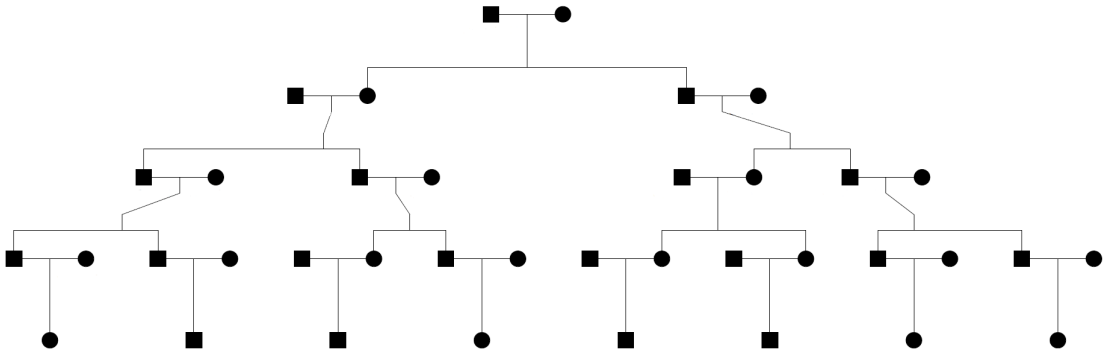

Figure 2.3: Example pedigree chart/tree for one family of simulated data.

Algorithm 5 prints the inferred relationships to a file for analysis of performance. It takes as input a name for the output file (that Algorithm 5 writes the results to), an id of a reference individual, and the sets for the first, second, and third degree relatives of that individual which are found by using Algorithm 4. If an individual is in the set of first degree relatives, then the algorithm prints the pair to file as a first degree relationship. If an individual is in the second degree set (but not in the first degree set), then the algorithm prints the pair to file as a second degree relationship. Lastly, if an individual is in the third degree set (but not in the second degree set), then the algorithm prints the pair to file as a third degree relationship. We used an additional python script to analyze the output files for accuracy. Table 2.1 lists the different parameter combinations that we used during testing.

Algorithm 5 Prints inferred relationships to file

Require: *set*1 refers to the first degree relatives
Require: *set*2 refers to the second degree relatives
Require: *set*3 refers to the third degree relatives
Require: *file* is the output file to write to
Require: *numIndivs* is the number of individuals
Require: *j* is the reference individual

```
1:  function PRINTMATCHES(set1, set2, set3, j, file)
2:      for each individual i in 1...numIndivs do
3:          inset1 = true if i is in set1
4:          inset2 = true if i is in set2
5:          inset3 = true if i is in set3
6:          if inset1 then
7:              Print ids of individuals j and i with degree 1 to file
8:          else if inset2 then
9:              Print ids of individuals j and i with degree 2 to file
10:         else if inset3 then
11:             Print ids of individuals j and i with degree 3 to file
12:         end if
13:     end for
14: end function
```

Superwindows	First	Second	Third	Window length(cM)
100	35	15	7	3.5
100	35	15	7	1
100	50	25	15	3.5
100	50	25	20	3.5
100	70	35	20	3.5
100	75	50	25	3.5
100	80	20	10	3.5
100	80	25	10	3.5
100	75	50	25	1
500	350	150	70	3.5
500	350	150	70	1
500	325	135	60	3.5
500	325	135	60	1
500	300	150	70	3.5
500	300	150	70	1
500	250	125	75	3.5
500	250	100	50	1
500	250	100	50	3.5
1000	350	150	70	3.5
1000	350	170	75	3.5
1000	360	160	70	3.5
1000	365	165	70	3.5
1000	400	200	60	3.5
1000	500	250	100	3.5
1000	800	200	100	3.5
1000	350	150	70	1
1000	365	165	70	1

Table 2.1: Different parameters used for degree prediction testing on a sample of 380 individuals.

2.4 Results

2.4.1 Without Genotyping Error

We examined the results shown in Figures 2.4 to 2.6 to determine the best threshold values when the method used intersection across window match sets for the

superwindow match sets. Figure 2.4 shows the results of inferring relatedness on the sample of 380 individuals using 100 superwindows. Figure 2.5 shows the results of inferring relatedness on the sample of 380 individuals using 500 superwindows. Figure 2.6 shows the results of inferring relatedness on the sample of 380 individuals using 1000 superwindows. For each superwindow count, each case uses a different combination of threshold percentages. We used some combinations across the superwindow counts. From analysis of these figures, it appears that using 500 superwindows performs the best across first, second, and third degree relatedness prediction. This may be due to the combinations that we used as they are more representative of the average thresholds than those used with 100 or 1000 superwindows. The threshold combinations used for 1000 superwindows are more on the lower side, which could explain the poorer performance. More discussion and testing of parameter values is covered later in this thesis when accounting for genotyping errors.

We examined the runtime of the algorithm using various parameter combinations as shown in Figure 2.7. On average, with the simulated data of 380 individuals, the algorithm takes roughly 8 seconds. To get a better understanding of our algorithm's efficiency, we used PLINK to infer relatedness on the simulated dataset and compared its runtime to that of the algorithm. Table 2.2 compares the runtime of PLINK to our algorithm with 500 superwindows, window length of 3.5 cM, and threshold values of 70%, 30%, and 14% for first, second, and third degrees when inferring relatedness on a sample of 380 individuals. The runtimes between the two methods are comparable. According to a benchmarking study by Ramstetter et al., PLINK is currently the fastest method for relatedness inference. [15]

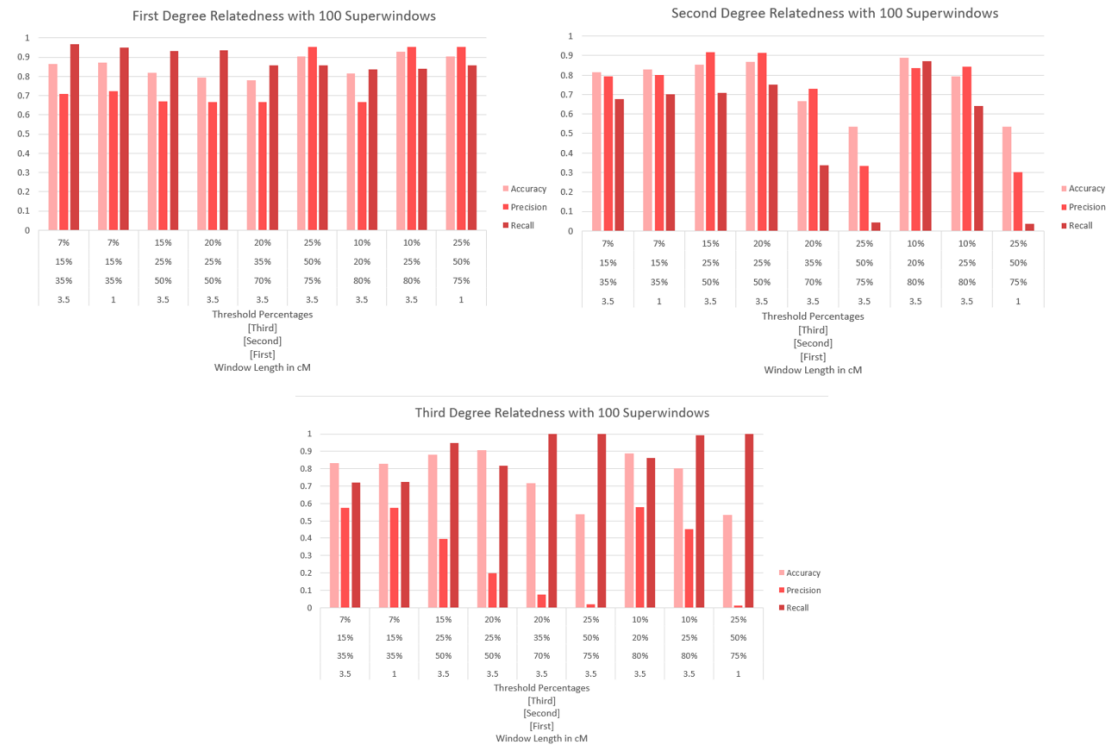

Figure 2.4: Accuracy, Precision, and Recall values of degree prediction using 100 superwindows on a sample of 380 individuals

Our Algorithm	**PLINK**
6.306 s	6.421 s

Table 2.2: Average runtimes of the developed algorithm and PLINK to infer relatedness on a sample size of 380 individuals.

In addition to runtime, we also compared the two algorithms for accuracy on relatedness inference. Since our algorithm only labels up to the third degree of relatedness, we ran PLINK's `--genome` command with the `--min 0.088` option since 8.8% is the lower bound for third degree relatedness used by Ramstetter et al.[15] We inferred the predicted degree of relatedness from PLINK by using the proportion IBD value ("the pi value") given by the `--genome` command. Each degree of relatedness corresponds to a range of IBD proportion values. For first degree relatedness, the IBD proportion is greater than 35.4%. For

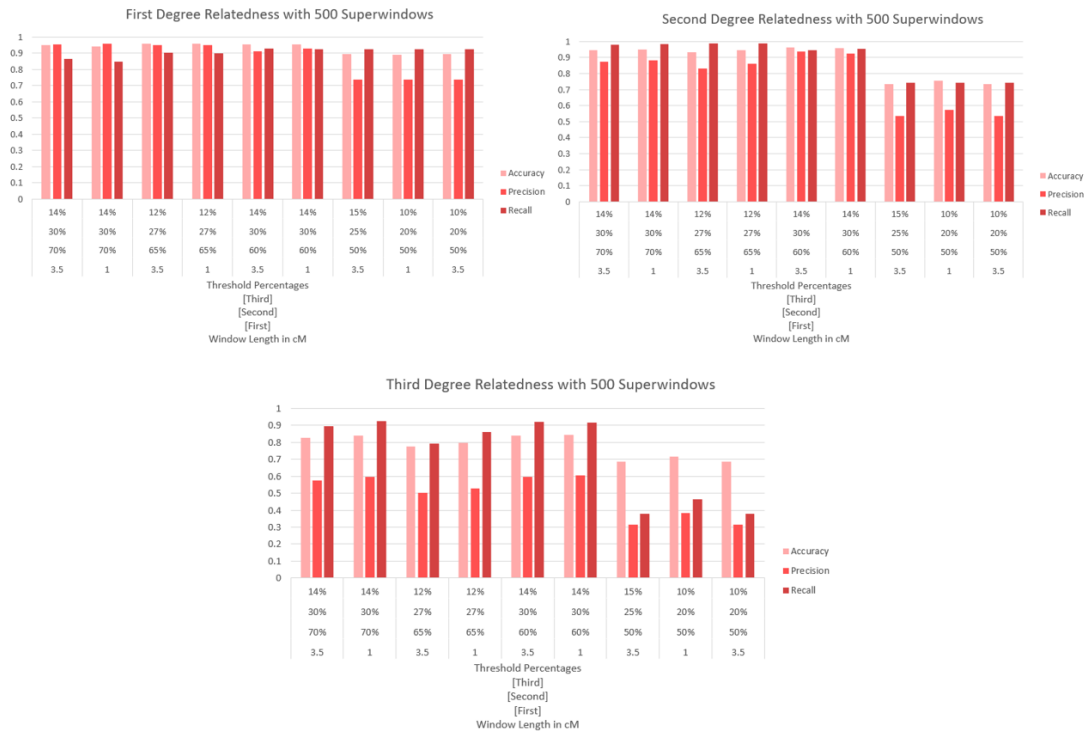

Figure 2.5: Accuracy, Precision, and Recall values of degree prediction using 500 superwindows on a sample of 380 individuals

second degree relatedness, the range of IBD proportion is from 17.7% to 35.4%. For third degree relatedness, the range is from 8.8% to 17.7%. The average accuracy values from using PLINK and our algorithm using 500 superwindows, window length of 3.5 cM, and threshold values of 70%, 30%, and 14% for first, second, and third degrees are shown in Table 2.3. Based on these values, PLINK outperforms our algorithm for inference of all degrees of relatedness. Further comparisons of the algorithm and PLINK for runtime and performance are done on larger sample sizes and with consideration of genotyping errors.

Figure 2.6: Accuracy, Precision, and Recall values of degree prediction using 1000 superwindows on a sample of 380 individuals

	First Degree	Second Degree	Third Degree
Our Algorithm	0.933366	0.879543	0.778915
PLINK	0.999436	0.977991	0.935102

Table 2.3: Average accuracy of degree prediction of the developed algorithm using 500 superwindows, window length of 3.5 cM, and threshold values of 70%, 30%, and 14% for first, second, and third degrees and PLINK on a sample of 380 individuals.

2.4.2 With Genotyping Error

With genotyping error incorporated, we determined the combination of threshold values that yielded the best results. Figures 2.8 to 2.16 summarize the accuracy, precision, and recall results for the various threshold combinations that we tested. We used the same combinations when using 100, 500, and 1000 superwindows on a sample size of 380 individuals with a window length of 3.5

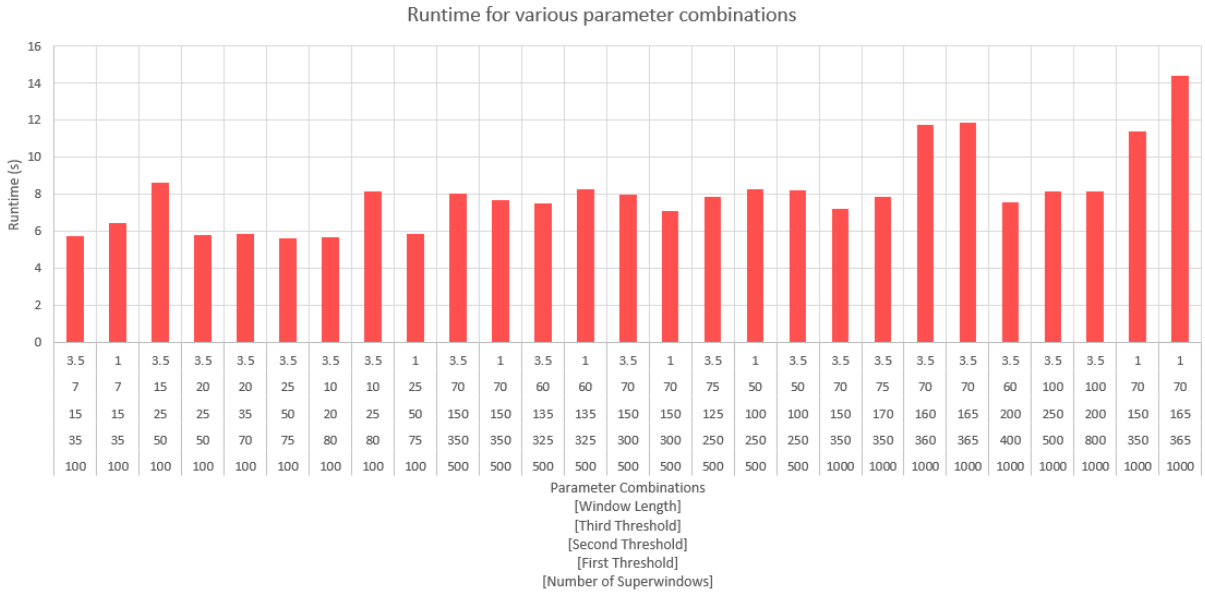

Figure 2.7: Runtimes of the algorithm without consideration of genotyping error under various parameter combinations

cM. We found that the performance values are not the same with differing superwindow counts for the same threshold combinations. For some threshold combinations, we found high accuracy but low precision and recall for degree prediction. This is because the selected threshold value is too high for correctly inferring the degree of relatedness. For example, the threshold combination of 80%, 90%, and 100% in Figure 2.9 results in high accuracy but very low precision and recall. The accuracy is high because this combination does not label non-second degree relatives as second degree. The precision and recall are low because this combination does not label many pairs as second degree relatives. From analyzing Figures 2.8 to 2.16, the best threshold combinations are as follows:

- With 100 superwindows, threshold values of 80%, 60%, and 40% for first, second, and third degrees were best.
- With 500 or 1000 superwindows, threshold values of 80%, 40%, and 30%

for first, second, and third degrees were best.

We used these threshold values in the next set of testing that looks at the influence of window length and sample size on relatedness inference efficiency and accuracy.

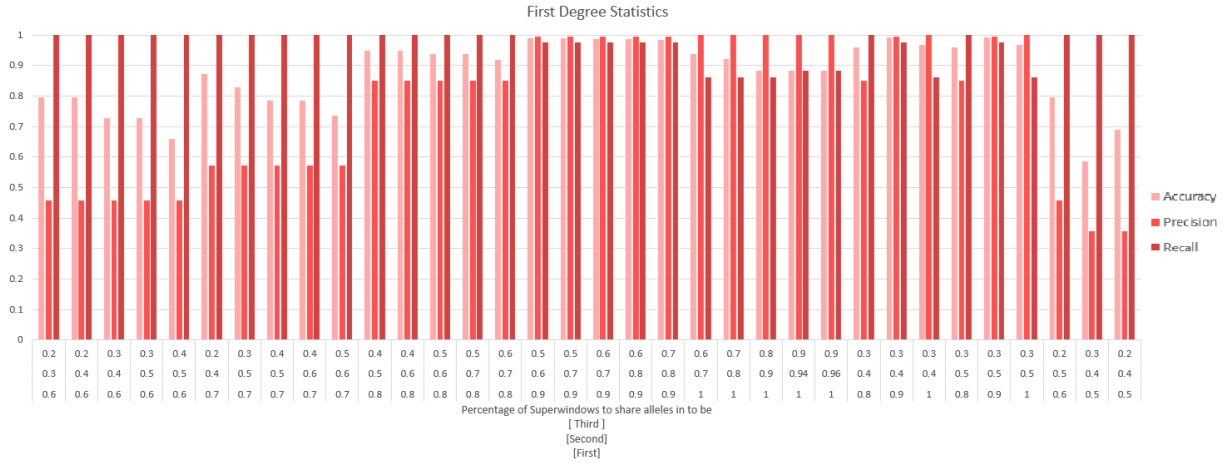

Figure 2.8: Accuracy, Precision, and Recall values of first degree prediction on 380 individuals using 100 superwindows and window length of 3.5cM and varying threshold values.

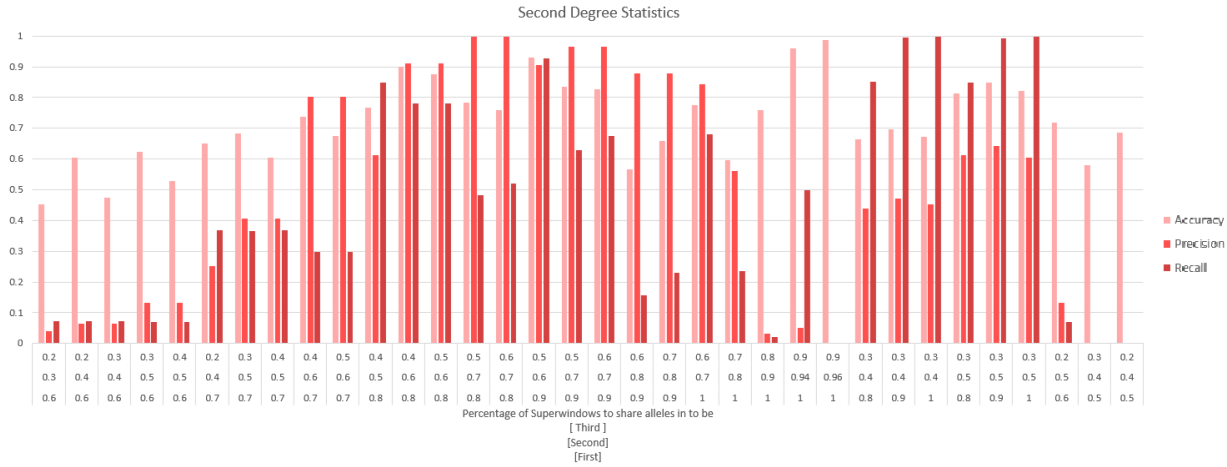

Figure 2.9: Accuracy, Precision, and Recall values of second degree prediction on 380 individuals using 100 superwindows and window length of 3.5cM and varying threshold values.

To study the influence of window length on the algorithm, we varied the

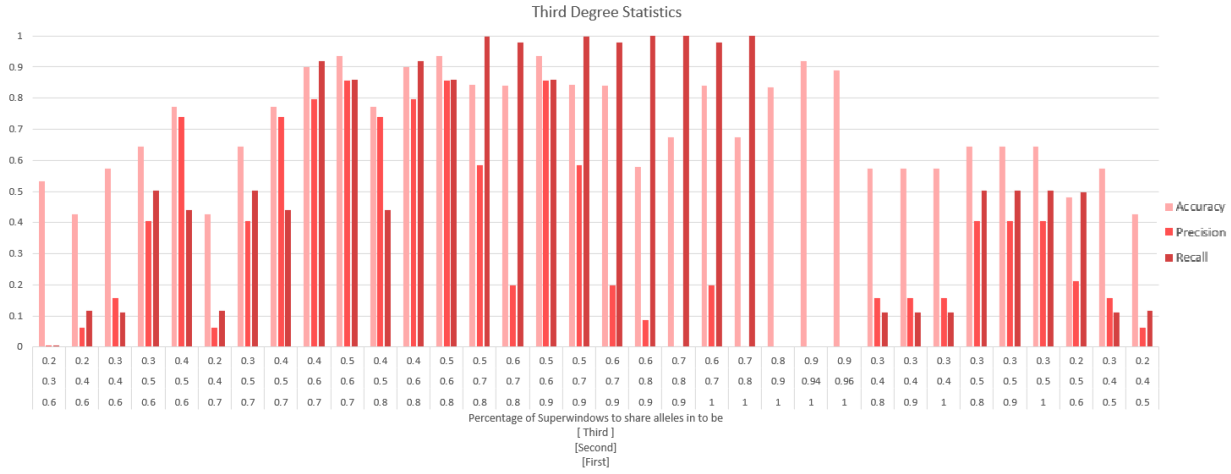

Figure 2.10: Accuracy, Precision, and Recall values of third degree prediction on 380 individuals using 100 superwindows and window length of 3.5cM and varying threshold values.

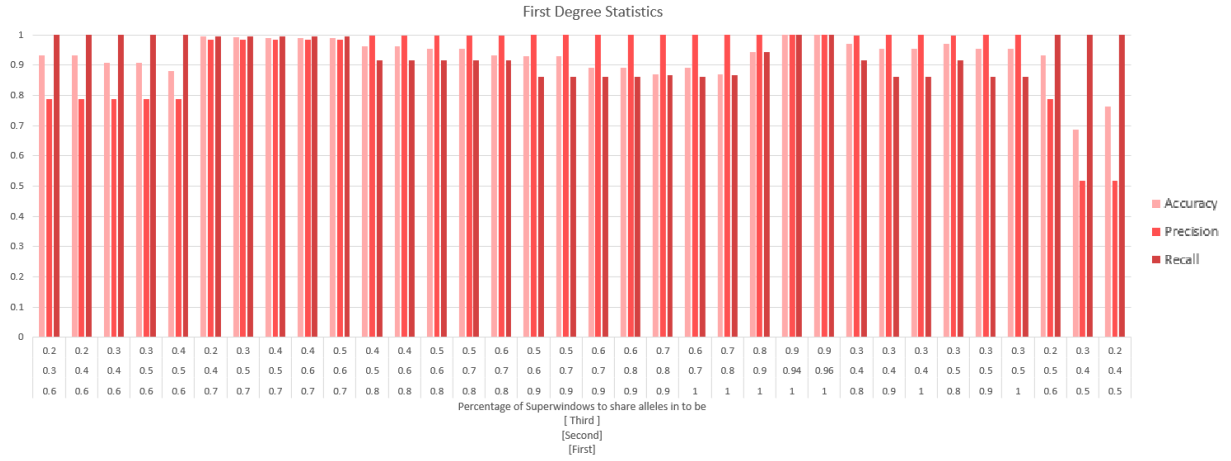

Figure 2.11: Accuracy, Precision, and Recall values of first degree prediction on 380 individuals using 500 superwindows and window length of 3.5cM and varying threshold values.

window length from 1 cM to 10.5 cM (varying by 0.5 cM) for a total of 20 different window lengths. Each superwindow count was run with these window lengths for varying sample sizes. The different sample sizes differed in the number of families present, where each family had the same pedigree structure as shown in Figure 2.3. The family counts of the different sample sizes were: 1, 5, 10, 15, 20, 50, 75, 100, and 130. These correspond to sample sizes of: 38, 190, 380,

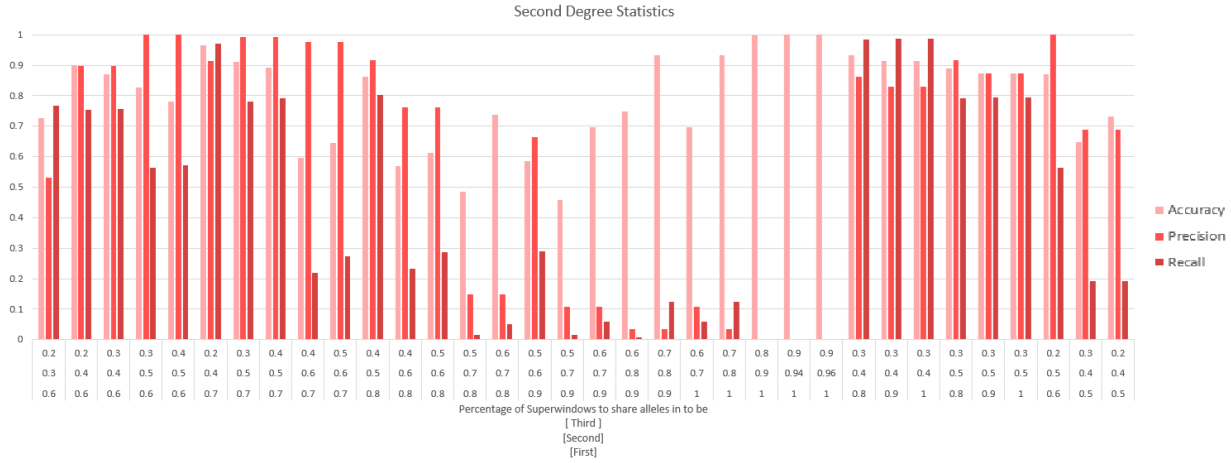

Figure 2.12: Accuracy, Precision, and Recall values of second degree prediction on 380 individuals using 500 superwindows and window length of 3.5cM and varying threshold values.

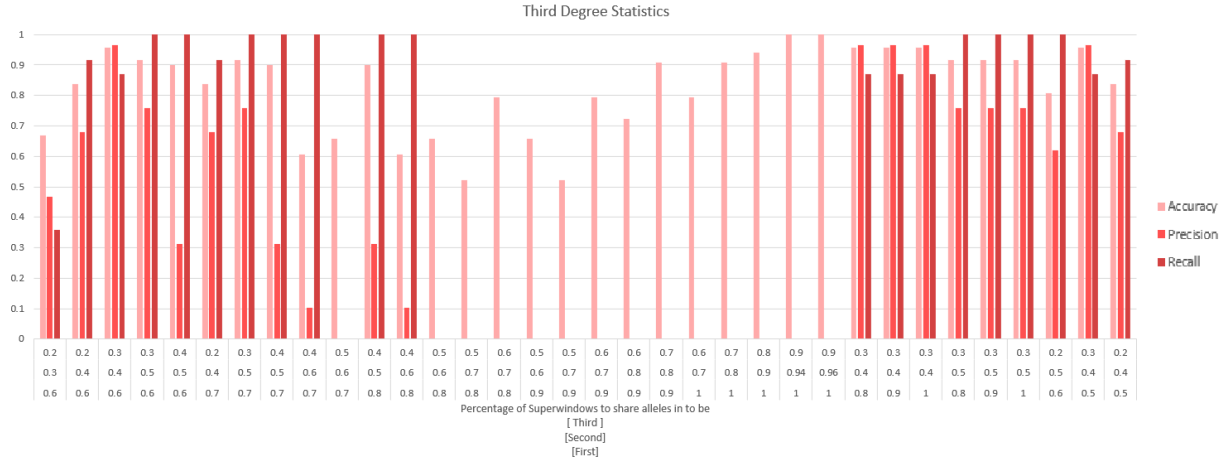

Figure 2.13: Accuracy, Precision, and Recall values of third degree prediction on 380 individuals using 500 superwindows and window length of 3.5cM and varying threshold values.

570, 760, 1900, 2850, 3800, and 4940 respectively.

Figures 2.17 to Figure 2.19 represent an example of the accuracy, precision, and recall plots under varying the window length. The plots provided are from using 1000 superwindows. As can be seen from the plots, after a certain window length, the values do not change. This is because the window length is

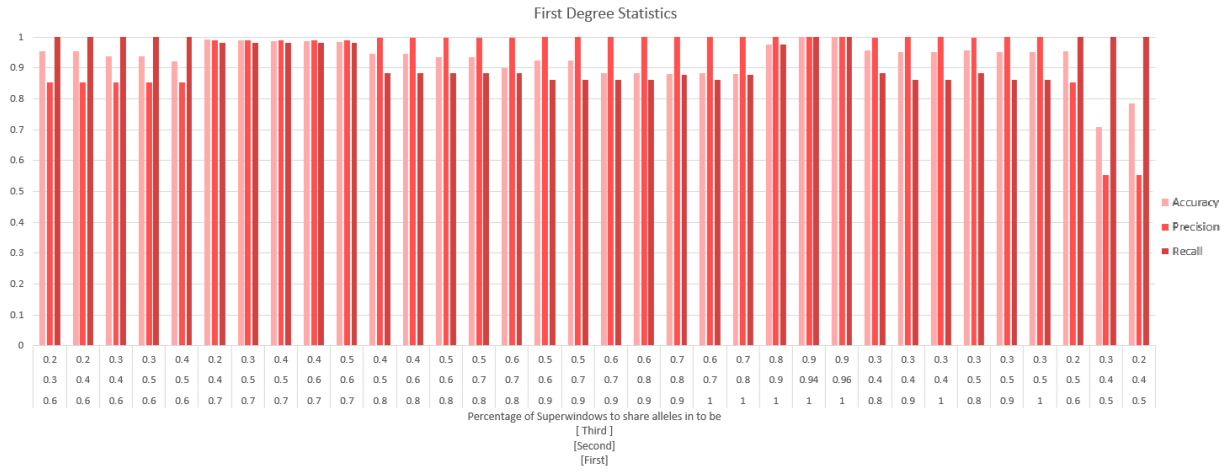

Figure 2.14: Accuracy, Precision, and Recall values of first degree prediction on 380 individuals using 1000 superwindows and window length of 3.5cM and varying threshold values.

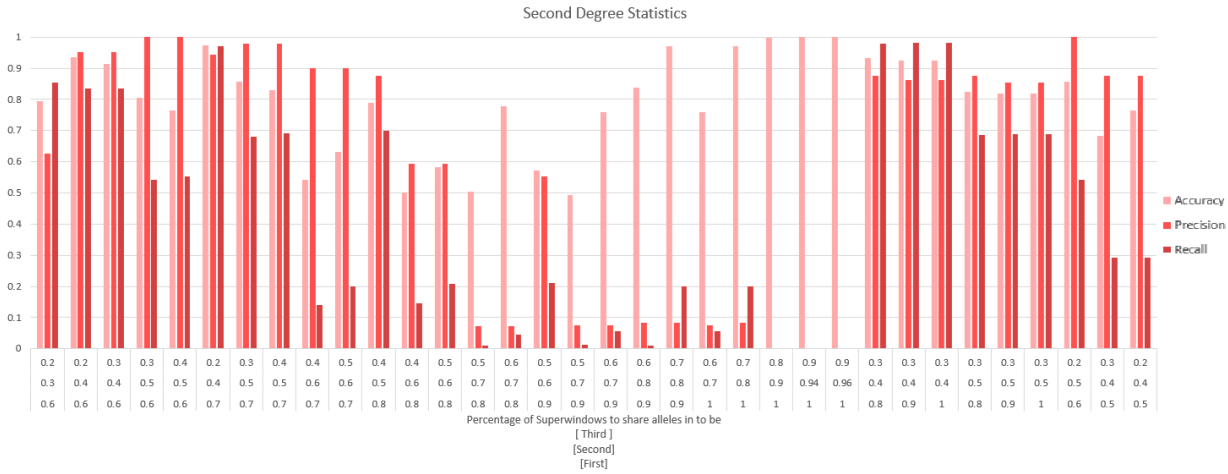

Figure 2.15: Accuracy, Precision, and Recall values of second degree prediction on 380 individuals using 1000 superwindows and window length of 3.5cM and varying threshold values.

specified to be much larger than the superwindow itself. In this case, the algorithm would treat the entire superwindow as one window. Similar trends are also seen when using 100 and 500 superwindows. On average, the algorithm has the best performance when there are about 3 windows within a superwindow. The number of windows within a superwindow depends on the window

Figure 2.16: Accuracy, Precision, and Recall values of third degree prediction on 380 individuals using 1000 superwindows and window length of 3.5cM and varying threshold values.

length and number of superwindows as described in section 2.2.1.

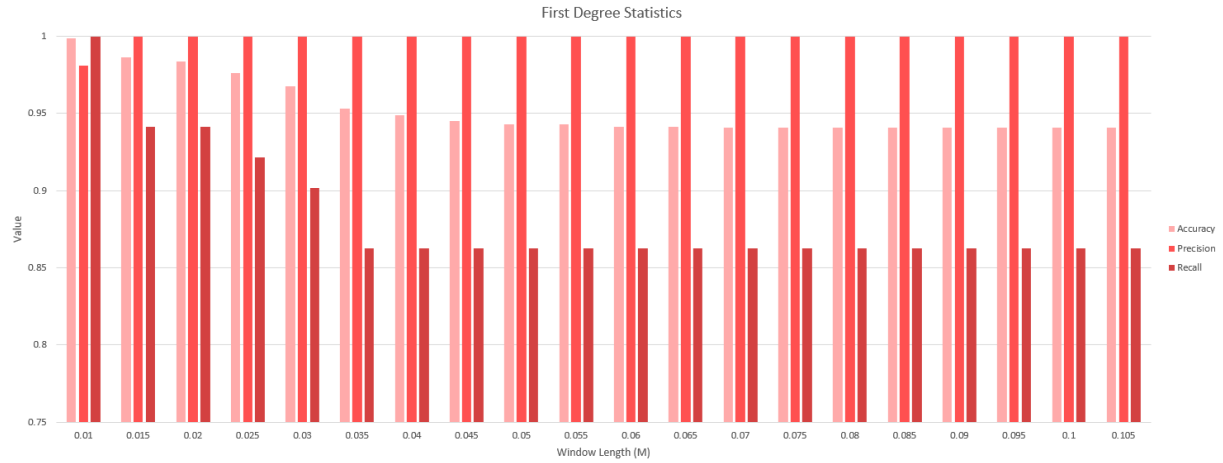

Figure 2.17: Accuracy, Precision, and Recall values of first degree prediction on a population of 38 using 1000 superwindows and a threshold of 80% for varying window lengths

As far as efficiency is concerned, the only parameters that impact the algorithm's runtime is the sample size and the number of superwindows. Figure 2.20 shows a comparison of the runtime of PLINK and the algorithm using 100, 500, and 1000 superwindows with varying sample sizes. As mentioned earlier,

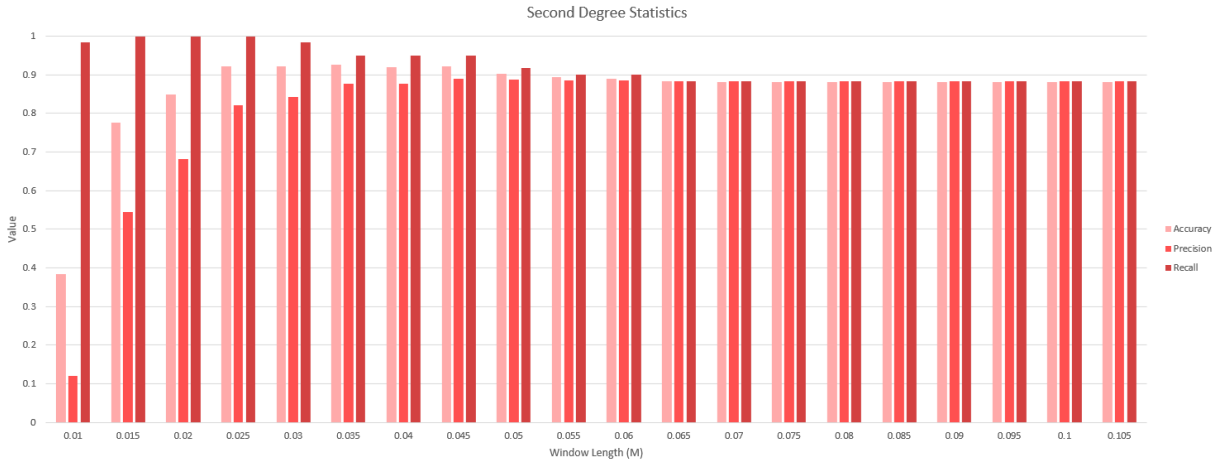

Figure 2.18: Accuracy, Precision, and Recall values of second degree prediction on a population of 38 using 1000 superwindows and thresholds of 40% for varying window lengths

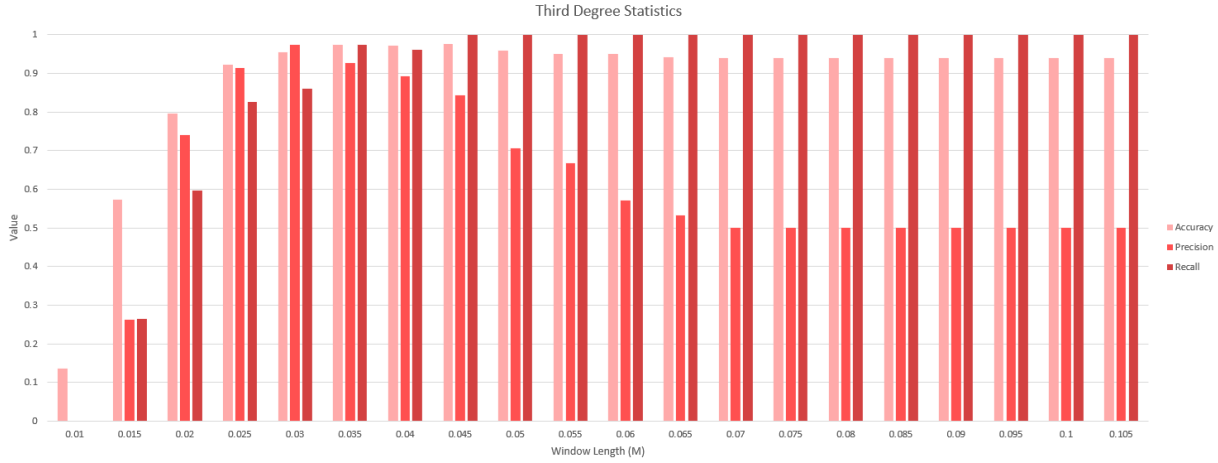

Figure 2.19: Accuracy, Precision, and Recall values of third degree prediction on a population of 38 using 1000 superwindows and thresholds of 30% for varying window lengths

we used threshold values of 80%, 60%, and 40% for first, second, and third degrees when using 100 superwindows and threshold values of 80%, 40%, and 30% when using 500 or 1000 superwindows. The algorithm was run with window lengths varying from 1 cM to 10.5 cM for each superwindow count on each sample size. We averaged the runtimes for each window length together. For smaller sample sizes, the runtimes of all algorithms appear to be comparable.

However, the algorithm performs much more efficiently than PLINK for larger sample sizes, with use of 100 and 500 superwindows outperforming use of 1000 superwindows. As expected, the runtime of the algorithm increases with sample size. However, on average, it is still faster than PLINK for larger sample sizes. The other parameters of window length and the threshold values are not significant factors of the runtime (data not shown).

We also compared PLINK to our algorithm for performance on degree prediction. We ran the algorithm using the same parameter combinations as in the runtime comparison above. We averaged the performance for different sample sizes together. The results are shown in Figure 2.21. We found that PLINK has an overall performance that is better in comparison to our algorithm. However, the results of our algorithm are roughly comparable to those of PLINK. We expected the performance of our algorithm to be below PLINK's performance since we focused on improving the runtime efficiency.

2.4.3 Future Directions

One possible extension of this study is to look into alternate approaches to finding the windows and superwindows. Currently, the algorithm finds a superwindow and then breaks it down into windows. An alternate approach would be to find windows first and then specify how many windows constitute a superwindow. Other expansions of this study include identification of relationships between more distantly related individuals such fourth and fifth degree relatives.

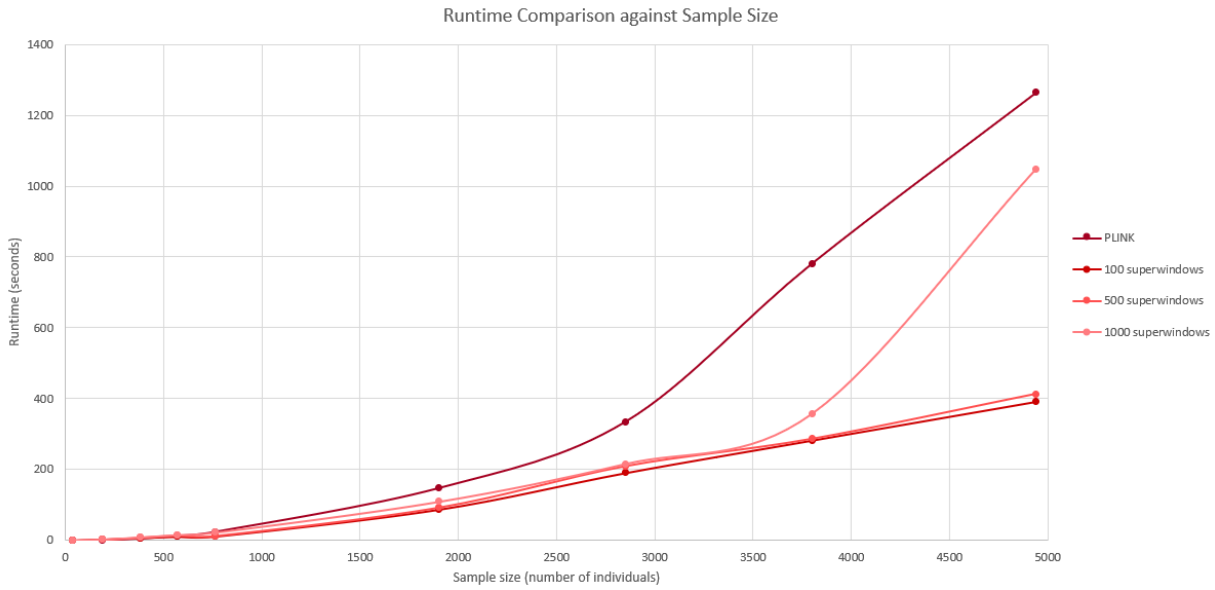

Figure 2.20: Average Runtime Comparison of PLINK against the algorithm for varying superwindow counts and sample sizes. With 100 superwindows, we used threshold values of 80%, 60%, and 40% for first, second, and third degrees. With 500 or 1000 superwindows, we used threshold values of 80%, 40%, and 30% for first, second, and third degrees. The algorithm was run with window lengths varying from 1 cM to 10.5 cM for each superwindow count on each sample size. We averaged the runtimes for each window length together.

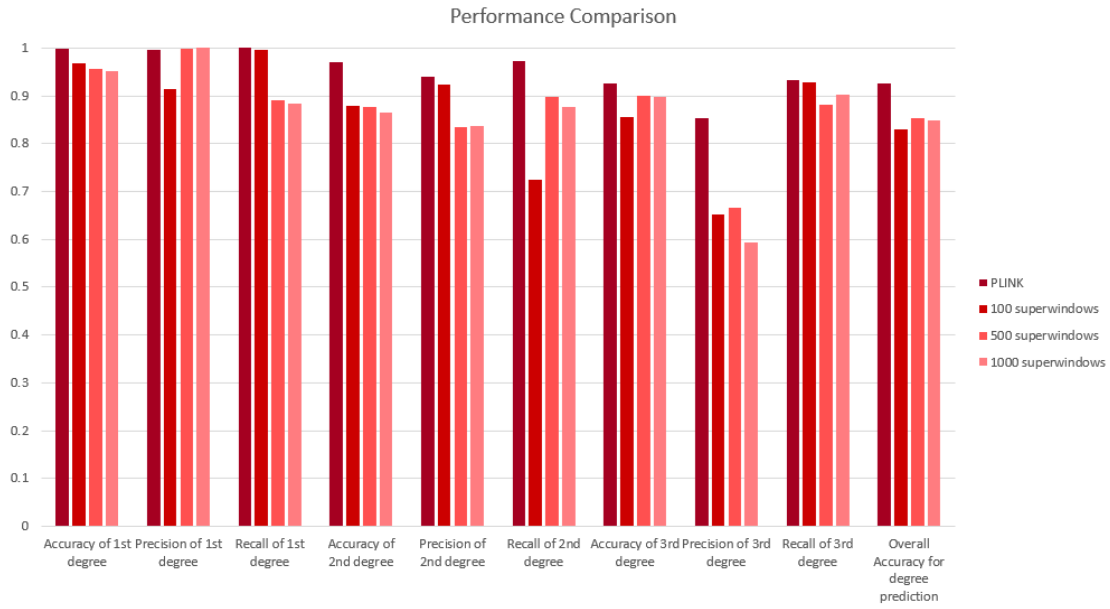

Figure 2.21: Performance Comparison of PLINK versus the algorithm for the same combinations used in the runtime comparison. The performance values for each sample size were averaged together.

REFERENCES

[1] "What is a gene?" *Genetics Home Reference. NIH* https://ghr.nlm.nih.gov/primer/basics/gene

[2] Hindorff LA, MacArthur J (European Bioinformatics Institute), Morales J (European Bioinformatics Institute), Junkins HA, Hall PN, Klemm AK, and Manolio TA. A Catalog of Published Genome-Wide Association Studies. https://www.genome.gov/gwastudies/

[3] Lewis CM, Knight J. Introduction to genetic association studies. *Cold Spring Harb Protoc.* 2012; 3: 297-306. http://cshprotocols.cshlp.org/content/2012/3/pdb.top068163.full

[4] Hu H., Roach J.C., Coon H., et al. A unified test of linkage analysis and rare-variant association for analysis of pedigree sequence data. *Nat. Biotechnol*. 2014; 32: 663669.

[5] International Human Genome Sequencing Consortium. Initial sequencing and analysis of the human genome.*Nature*, 15 February 2001; 409: 860921. https://doi.org/10.1038/35057062

[6] Gibbs R.A., Belmont J.W., Hardenbol P., Willis T.D., Yu F. et al. The international HapMap project. *Nature*, 2003; 426(6968):789-796.

[7] 1000 Genomes Project Consortium et al. A global reference for human genetic variation. *Nature*, 2015; 526(7571):68-74.

[8] Ungerleider N. "Ancestors, Inc.: Inside The Remarkable Rise Of The Genealogy Industry" *Fast Company* July 15, 2015. https://www.fastcompany.com/3048513/ancestors-inc-inside-the-remarkable-rise-of-the-genealogy-industry

[9] O'Connell J.R. and Weeks D.E. PedCheck: a program for identifcation of genotype incompatibilities in linkage analysis. *American Journal of Human Genetics* 1998; 63(1): 259-266

[10] Bochud M. Estimating heritability from nuclear family and pedigree data. *Methods Mol Biol* 2012; 850: 171-86. https://doi.org/10.1007/978-1-61779-555-8_10.

[11] Paul DB, Spencer HG. "It's Ok, We're Not Cousins by Blood": The Cousin Marriage Controversy in Historical Perspective. *PLoS Biol* 2008 Dec: 6(12). https://doi.org/10.1371/journal.pbio.0060320

[12] Jobling MA, Gill P. Encoded evidence: DNA in forensic analysis. *Nature Reviews Genetics* 2004; 5(10): 739-751.

[13] Purcell S., Neale B., Todd-Brown K., Thomas L., Ferreira M., Bender D., et al. Plink: A tool set for whole-genome association and population-based linkage analyses. *Am. J. Hum. Genet.* 2007; 81: 559-575.

[14] Chang C., Chow C., Tellier L., Vattikuti S., Purcell S., Lee J. Second-generation PLINK: rising to the challenge of larger and richer datasets, *GigaScience*, 1 December 2015; 4(1):116, https://doi.org/10.1186/s13742-015-0047-8

[15] Ramstetter M.D., Dyer T.D., Lehman D.M., Curran J.E., Duggirala R., Blangero J., Mezey J.G., and Williams A.L. Benchmarking Relatedness Inference Methods with Genome-Wide Data from Thousands of Relatives. *Genetics* September 1, 2017; 207(1): 75-82. https://doi.org/10.1534/genetics.117.1122

[16] Frazer, K. A., Ballinger, D. G., Cox, D. R., Hinds, D. A., Stuve, L. L., Gibbs, R. A., Stewart, J. A second generation human haplotype map of over 3.1 million SNPs. *Nature*, 2007; 449(7164): 851-861. https://doi.org/10.1038/nature06258

[17] Ramstetter M.D., Shenoy S., Dyer T.D., Lehman D.M., Curran J.E., Duggirala R., Blangero J., Mezey J.G., and Williams A.L. Inferring identical by descent sharing of sample ancestors promotes high resolution relative detection. (2018) https://doi.org/10.1101/243048

[18] International Multiple Sclerosis Genetics Consortium, Wellcome Trust Case Control Consortium 2, et al. Genetic risk and a primary role for cell-mediated immune mechanisms in multiple sclerosis. *Nature*, 2011; 476(7359): 214-219